A **COMPILER DESIGN LAB** Project

Submitted to

Institute Of Engineering & Management, Kolkata

Towards fulfilment for The

Bachelor of Technology

in

INFORMATION TECHNOLOGY

Under Guidance of: – Prof. **Subhabrata Sengupta** &

Prof. Pulak Baral

Submitted by –

RAVI RANJAN (128)

VISHAL MANDAL (123)

SUMIT KUMAR SAXENA(137)

ADITYA PAUL(127)

ARPAN GOSWAMI (134)

CERTIFICATE OF APPROVAL

This is to certify that the work embodied in this project entitled **Compiler Design Lab** submitted by our group to the Department of Information Technology, have been carried out under my direct supervision and guidance. The project work has been prepared as per the regulations of Institute of Engineering and Management and I strongly recommend that this project workbe accepted in fulfillment of the minor Compiler Design Lab project.

Supervisors

Prof. Subhabrata Sengupta
Dept. of Information Technology

Prof. Pulak Baral
Dept. of Information Technology

Experiment No. :- 1

Aim:-

To identify whether a given line is Comment Line or not.

Theory :-

The goal of this experiment is to identify whether a given line is a comment line or not.

COMMENT LINE

In programming, comments are hints that a programmer can add to make their code easier to read and understand.

TYPES OF COMMENT LINE

Types of comments in programs:

```
Single Line Comment: Comments preceded by a Double Slash (‘//’)
Multi-line Comment: Comments starting with (‘/*’) and ending with (‘*/’).
```

input:-

```
#include <stdio.h>
int main() {
  // print Hello World to the screen
  printf("Hello World");
  return 0;
}
```

output:-

```
Hello World
```

1. Single-line Comments in C

In C, a single line comment starts with //. It starts and ends in the same line. For example,

```
#include <stdio.h>
int main() {

  // create integer variable
  int age = 25;

  // print the age variable
  printf("Age: %d", age);

  return 0;
}
```

output:-

```
Age: 25
```

2. Multi-line Comments in C

In C programming, there is another type of comment that allows us to comment on multiple lines at once, they are multi-line comments. To write multi-line comments, we use the /*....*/ symbol. For example,

```
#include <stdio.h>
int main() {

  // create integer variable
  int age = 25;

  // print the age variable
  printf("Age: %d", age);

  return 0;
}
```

output:-

```
Age: 25
```

Procedure:-

The simulator tab would:

- Please Enter Any Line in Text-Box Area.
- Enter Submit .
- It will Check if 1st position and 2nd position of a String is equal to "/" then "IT IS COMMENT LINE".
- If above is not True then Check if 1st position and last position of String is "/" and 2nd position and 2nd-last position of string is "*", then "IT IS COMMENT LINE".
- If both are not True then "NOT A COMMENT LINE".
- Press clear button for clear Text Box.

Simulation:-

References:-

- https://www.javatpoint.com/comments-in-c

Experiment No. :- 2

Aim:-

Design a lexical analyzer for given language and the lexical analyzer should ignore redundant spaces, tabs and new lines. It should also ignore comments, the length of the user input string can be user specific. From this particular statement count total number of individual token.

Theory:-

- Lexical analysis is the very first phase in the compiler design.
- Lexemes and Tokens are the sequence of characters that are included in the source program according to the matching pattern of a token.
- Lexical analyzer is implemented to scan the entire source code of the program.
- Lexical analyzer helps to identify token into the symbol table .
- Removes one character from the remaining input is useful Error recovery method.
- Lexical Analyser scan the input program while parser perform syntax analysis.
- It eases the process of lexical analysis and the syntax analysis by eliminating unwanted tokens

Basic Terminologies:-

What's a lexeme?

A lexeme is a sequence of characters that are included in the source program according to the matching pattern of a token. It is nothing but an instance of a token.

What's a token?

Tokens in compiler design are the sequence of characters which represents a unit of information in the source program.

Roles of the Lexical analyzer:-

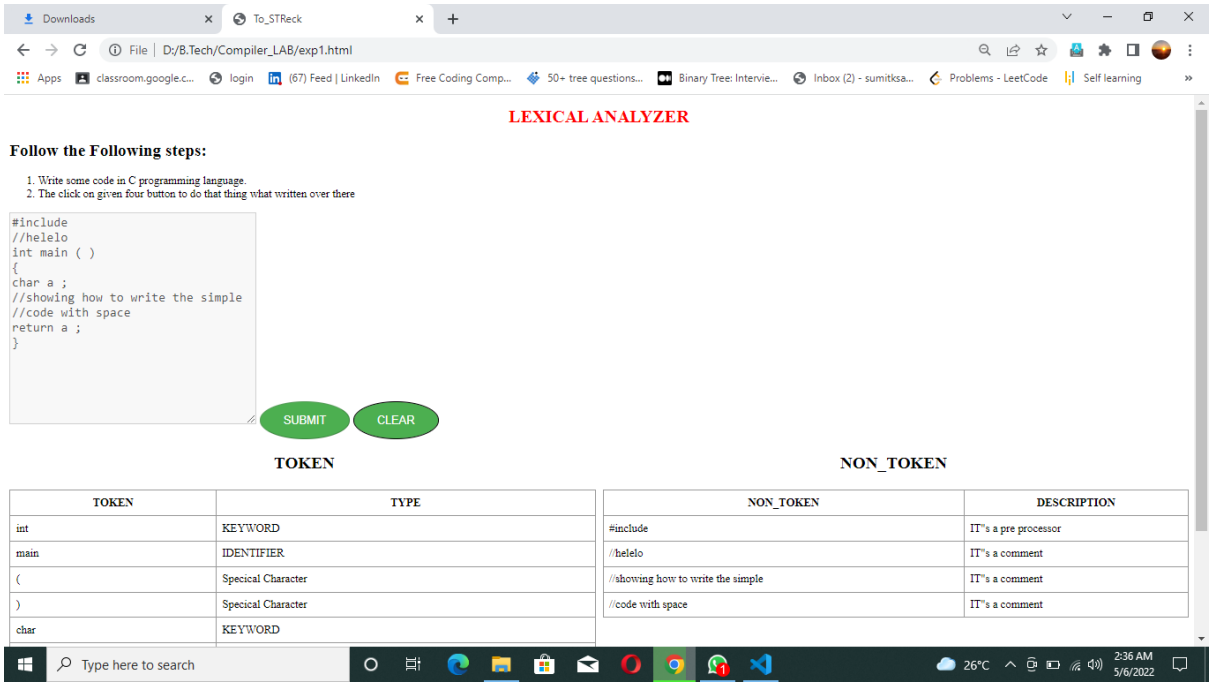
Lexical analyzer performs below given tasks:

- Helps to identify token into the symbol table
- Removes white spaces and comments from the source program
- Correlates error messages with the source program
- Helps you to expands the macros if it is found in the source program
- Read input characters from the source program

Procedure:-

1. Write some code in C programming language (NOTE:- Give Space every new word/character/operator).
2. Then click on given Submit button.
3. It will generate Tokens, identifiers and keywords list in the page.

Simulation:-



Reference:-

- https://www.guru99.com/compiler-design-lexical-analysis.html

Experiment No. :- 3

Aim:-

To recognize strings under ‘a*’,’a*b+’,abb.

Theory:-

set of **productions** (P). The productions of a grammar specify the manner in which the terminals and non-terminals can be combined to form strings. Each production consists of a **non-terminal** called the left side of the production, an arrow, and a sequence of tokens and/or **on- terminals**, called the right side of the production.

a sequence of production rules, in order to get the input string. During parsing, we take two decisions for some sentential form of input:

- Deciding the non-terminal which is to be replaced.
- Deciding the production rule, by which, the non-terminal will be replaced.

Procedure:-

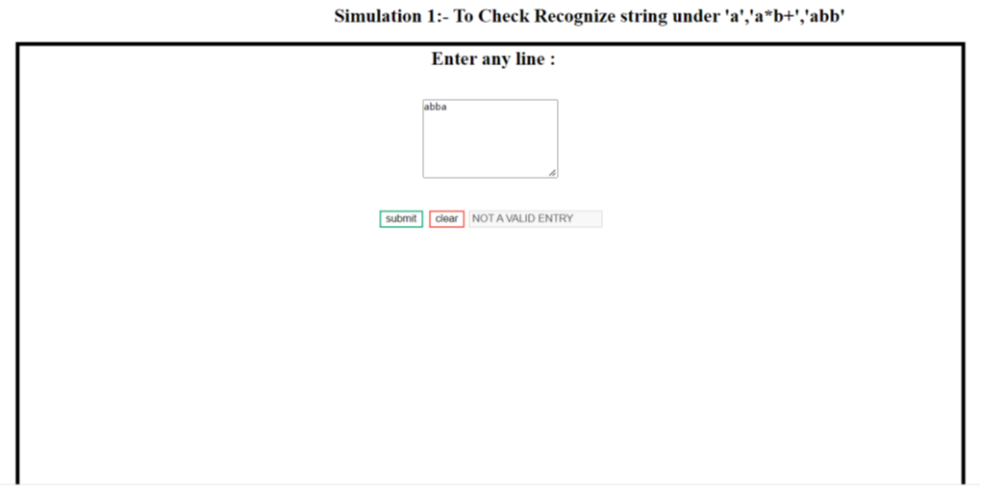
- Enter Expression in Text-box.
- Click Button Clear for clear the text box and result box.
- Click Button submit and then .

Inside Algorithm will work:-

- Get String/Expression from user.
- Take a counter variable c and initialize 1.
- Check 1st position is not ‘a’ then break And print “NOT A VALID ENTRY” else next step.
- Now create a loop for check every character , if character is ‘b’ then c=0.
- Elseif character is ‘a’ and c==0 then print “NOT A VALID ENTRY” and break.
- And one more condition, if character is neither ‘a’ nor ‘b’ then print “NOT A VALID ENTRY”.

- Outside loop we have to check whether loop reached last character or not if it reached then print ”VALID ENTRY”.

Simulation:-



References:-

- https://www.tutorialspoint.com/compiler_design/compiler_design_syntax_analysis.htm#:~:text=The%20productions%20of%20a%20grammar,right%20side%20of%20the%20production.

Experiment No. :- 4

Aim:-

To check whether an given identifier is valid or not.

Theory:-

Identifier:-

C identifiers represent the name in the C program, for example, variables, functions, arrays, structures, unions, labels, etc. An identifier can be composed of letters such as uppercase, lowercase letters, underscore, digits, but the starting letter should be either an alphabet or an underscore.

Rules for constructing C identifiers

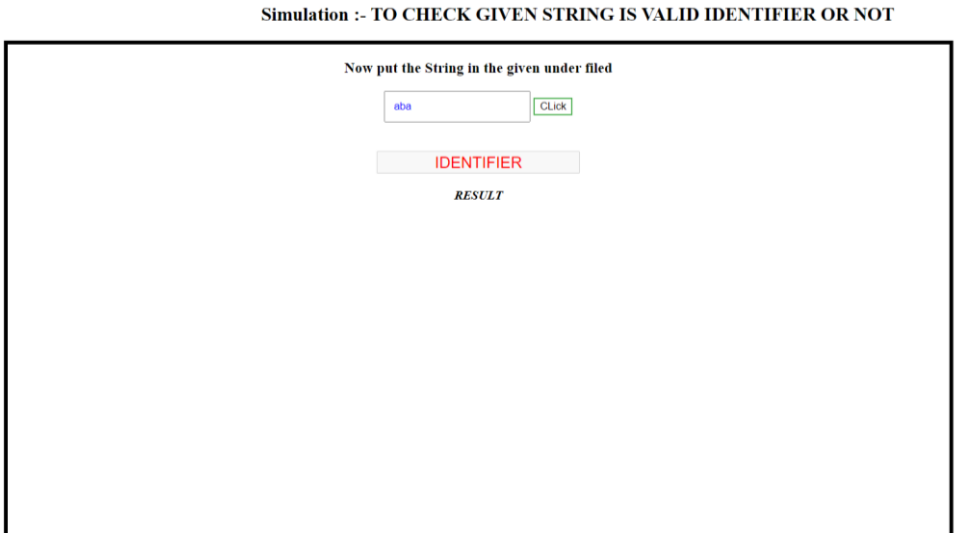
- The first character of an identifier should be either an alphabet or an underscore, and then it can be followed by any of the character, digit, or underscore.
- It should not begin with any numerical digit.
- In identifiers, both uppercase and lowercase letters are distinct. Therefore, we can say that identifiers are case sensitive.
- Commas or blank spaces cannot be specified within an identifier.
- Keywords cannot be represented as an identifier.
- The length of the identifiers should not be more than 31 characters.
- Identifiers should be written in such a way that it is meaningful, short, and easy to read

Procedure:-

1. Read the given input string from used by text-box.
2. Then Click on “Click” button .
3. It will Check the initial character of the string is numerical or any special character except ‘_’ then print it is not a valid identifier.

4. Otherwise print it as valid identifier if remaining characters of string doesn't contains any special characters except '_'.

Simulation:-



Reference:-

- https://www.javatpoint.com/c-identifiers#:~:text=C%20identifiers%20represent%20the%20name,an%20alphabet%20or%20an%20underscore.

Experiment No. :- 5

Aim:-

To implementing Shift Reduce Parsing

Theory:-

Shift Reduce parser attempts for the construction of parse in a similar manner as done in bottom-up parsing i.e. the parse tree is constructed from leaves(bottom) to the root(up). A more general form of the shift-reduce parser is the LR parser.
This parser requires some data structures i.e.

- An input buffer for storing the input string.
- A stack for storing and accessing the production rules

Basic Operations –

- **Shift:** This involves moving symbols from the input buffer onto the stack.
- **Reduce:** If the handle appears on top of the stack then, its reduction by using appropriate production rule is done i.e. RHS of a production rule is popped out of a stack and LHS of a production rule is pushed onto the stack.
- **Accept:** If only the start symbol is present in the stack and the input buffer is empty then, the parsing action is called accept. When accepted action is obtained, it is means successful parsing is done.
- **Error:** This is the situation in which the parser can neither perform shift action nor reduce action and not even accept action.

Procedure:-

1. Put 1 or 2 or 3 any in very first text box.
2. Now when we put you are able to see the production in the another text area there.
3. Last just select given string as there you can see the dropdown
4. And click to submit
5. You able to see full way of **Shift Reduce Parse.**

Simulation:-

References:-

- https://www.geeksforgeeks.org/shift-reduce-parser-compiler/